(TELO546)

Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2330707 A1 2002/07/12

(21) 2 330 707

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(22) Date de dépôt/Filing Date: 2001/01/12

(41) Mise à la disp. pub./Open to Public Insp.: 2002/07/12

(51) Cl.Int.^{7}/Int.Cl.^{7} H04L 29/02

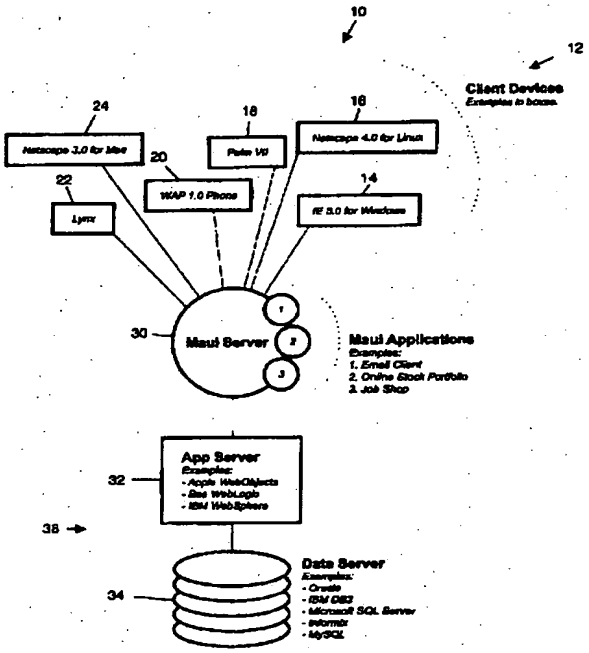
(71) Demandeur/Applicant:
BITMOVERS SOFTWARE, INC., CA

(72) Inventeurs/Inventors:
WOJTOWICZ, IAN S., CA;
GIBSON, PATRICK J., CA;
KNIGHT, CHRISTOPHER P., CA;
LAPOINTE, DAVID G. P., CA

(74) Agent: VERMETTE & CO.

(54) Titre : METHODE ET APPAREIL D'ANALYSE DES INTERACTIONS DE RESEAU DEMANDE-REPONSE INDEPENDANTES DES DISPOSITIFS

(54) Title: METHOD AND APPARATUS FOR ABSTRACTING DEVICE INDEPENDENT REQUEST-RESPONSE NETWORK INTERACTIONS



(57) Abrégé/Abstract:
The present invention relates to a method and apparatus for abstracting device-independent request-response network interactions.

ABSTRACT

The present invention relates to a method and apparatus for abstracting device-independent request-response network interactions.

METHOD AND APPARATUS FOR ABSTRACTING DEVICE INDEPENDENT REQUEST-RESPONSE NETWORK INTERACTIONS

FIELD

The present invention is directed to a method and apparatus for abstracting device-independent request-response network interactions using a high-level application programming interface.

BACKGROUND OF THE INVENTION

Deploying applications on the Internet with its widely varying protocols for transport and communications has become a daunting task. Much effort must be expended to rework portions of the same applications in order to be compatible with different client hardware and software configurations. Additionally, many recently developed technologies and tools for the Internet are attempts to bridge the fundamentally incompatible concepts and patterns upon which stand-alone client applications are built.

Further, since the fundamental concepts and patterns upon which stand-alone client applications are built cannot be directly represented or supported via most Internet protocols, many recent Internet technologies and tools have been developed as attempts to smooth over and compensate for the basic incompatibility.

There are many products for developers to choose from when writing applications for use over the Internet. However, these applications do not address the challenege of how a single application, in a single code base, can be presented to a wide variety of client devices and protocols.

In general, device independent request-response network interactions refer to data exchanges between a client device and a server device over a previously agreed to non-persistent communications medium. Examples of device independent request-response network interactions include communication between an HTTP client and a server via a series of non-persistent TCP/IP socket connections and the communication between a mobile phone and a network server using SMS message packets.

Presently, developers of network-based software applications, such as those applications for the World Wide Web or for mobile telephones, are required to implement their software by dealing directly with the client-server interactions inherent in these technologies. Programmers must directly implement rules for handling each and every data communication request sent from the client device, and they must provide rules for sending response communications for each individual request. This procedure is very cumbersome and inefficient as programmers as forced to spend their time writing software to handle network interactions instead of spending their time writing software to solve the problem at hand.

One of the difficulties recognized when programming device independent software applications is that Client Software Programs are often designed to work only with specific Server programs and vice-versa.

As a result of these difficulties, there is a need for a method and apparatus that allows developers to develop device-independent network based software applications without directly dealing with low-level request-response interactions.

Furthermore, there is a need for a technology that allows programmers to create a single application in a single code base that can be presented to a variety of client devices and protocols.

SUMMARY OF THE INVENTION

According to the invention there is provided a method and apparatus for abstracting device independent request-response network interactions. The method of request-response network interactions takes place over network. In general, the network will be comprised of Client Devices, a Maui Server, an Application Server and a Data Server.

Client Devices

Client devices are that class of device that contact and obtain data from a server. In most cases, the Client Device will be running a Client Program that performs the request-response interactions with a server. Examples of Client Devices include mobile phones, PDAs and personal computers.

The Client Devices communicate with the Application Server and the Data Server through the Maui Server. It is important to note that the Application Server, Data Server and Maui Server may be a single machine or a plurality of machines distributed throughout the network.

Maui Server

The Maui Server is the principal device for carrying out the method for abstracting device independent request-response network interactions. The Maui Server provides a conduit for the transfer of information (requests and responses) between Client Devices. The Maui Server is comprised of three main components: the Client Engine, The

Server Engine and the Application Environment. Together, the Client Engine and The Server Engine form the Maui Engine.

The Client Engine consists of Client Drivers, Compositors, Layout Managers and Client Components. The Client Drivers act as a translator between Client Devices and the programs that access the Client Devices. The Layout Managers manage the Graphical User Interface (GUI) components for the different types of Client Devices. The Compositors are platform specific components relied upon to provide the rendering for different Client Devices and the underlying protocols. The Client Components are generic GUI components such as windows, buttons and scroll bars.

The Server Engine consists of Server Components and Server Drivers. The Server Drivers act as a translator between Servers and the programs that access the Servers. The Server Components are visual components such as generic GUI components, which include windows, buttons and scroll bars, and non-visual components such as tables.

The Application Environment is the Application Program Interface (API) providing an interface to programmers to fully utilize the Maui Engine. The Application Environment also includes three levels of access: basic, advanced and expert depending on the skill level of the programmer.

The Application Server and the Data Server are standard network components and thus will not be described in any further detail.

The method for abstracting device independent request-response network interactions involves the translation of client centric request data packages into server centric Event objects and correspondingly the translation of server

centric Event objects into client centric response data packages.

The objective of the method for abstracting device independent request-response network interactions is to provide software developers with a standard mechanism of integrating object-oriented server software components with client software. Therefore, programmers are allowed software applications to be written as though they were desktop applications thereby simplifying the programming of networked applications. This invention fully abstracts all knowledge of the underlying protocol away from the application developer thus eliminating the need to deal directly with the communications protocol and allowing the application to be portable over a variety of protocols.

Using the Maui Engine, the software objects that represent GUI components may be assembled in collections to describe the layout of a software application's screen interface. The rendering process is dynamic allowing the output by the system to be customized to the Client Devices specific system requirements. It also provides the further advantages of allowing the development of web software applications without any knowledge of HTML or graphic design.

Other objects and advantages of the invention will become clear from the following detailed description of the preferred embodiment, which is presented by way of illustration only and without limiting the scope of the invention to the details thereof.

BRIEF DESCRIPTION OF THE DRAWINGS

Further features and advantages will be apparent from the following detailed description, given by way of example,

of a preferred embodiment taken in conjunction with the accompanying drawings, wherein:

Fig. 1 is a schematic block diagram of a sample network; and

Fig. 2 is a schematic block diagram of the components of the Maui Server.

DETAILED DESCRIPTION

Throughout the figures, like elements are indicated by like reference numbers.

Referring to **Fig. 1**, a typical network **10** is depicted having a plurality of client devices **12**, a Maui server **30**, an application server **32** and a data server **34**. The application server **32** and the data server **34** can be collective referred to as the server **36**.

The client devices **12** are that class of device that enters into request-response interactions with the server **36**. In most cases, the client devices **12** will be running a client program that performs the request-response interactions with the server **36**. Examples of client devices **12**, include, but are not limited to: client **14** that is a computer running the client program Microsoft Internet Explorer 5.0 for Windows; client **16** is a computer running the client program Netscape 4.0 for Linux; client **18** is a Personal Digital Assistant (PDA), client **20** is a mobile phone running WAP 1.0 as client program; client **22** is a personal computer running Lynx as a client program; and client **24** is a computer running Netscape 3.0 for the Mac as a client program.

It is clear from the example listed above, that the there are numerous possible client devices and client programs that may be utilized within the framework outlined herein.

The Maui server **30** provides a conduit between the client devices **12** and the server **36**. It is important to note that the Maui server **30**, the application server **32** and the data server **34** may be a single computer or a plurality of computer distributed throughout a network. The application server **32** will be running any software capable of developing and deploying Internet, intranet, extranet and e-commerce applications. Further, the software running on the application server **32** should have the ability to utilize Java servlets, JavaServer Pages, XML and Enterprise JavaBeans (EJB) components. The data server **34** will be running any data base program capable managing the required data. For example, the data server may be running Oracle™, IBM DB2™, or Microsoft SQL Server™. The application server **32** and the data server **34** are standard network components and thus will not be described in any further detail.

Referring to **Fig. 2**, the Maui server **30** is the principal device for carrying out the method for abstracting device independent request-response network interactions. The Maui server **30** is comprised of three main components: the client engine **40**, the server engine **50** and the application environment **60**.

The client engine **40** is further comprised of client drivers **42**, compositors **44**, layout managers **46** and client components **48**.

The client drivers **42** act as a translator between client devices **12** and the programs that communicate with the

client devices **12**. Each of the client devices **12** requires a different client driver. The driver accepts generic commands from a communicating program and translates them into specialized commands for the particular client device. The use of drivers provides the advantage of allowing a high-level programming interface for access to the devices. The developer is not required to delve into the intricacies of communicating with every one of the client devices **12**.

The compositors **44** are platform specific components relied upon to provide the rendering for the client devices **12** and the underlying client programs and protocols. The compositors integrate all of the graphics and text in a manner to be displayed on a particular client device. As the compositors are platform specific, a compositor will be required for each of the different types pf protocols and client devices **12**.

The layout managers **46** determine where the GUI components will be placed in relation to one another when displayed on the client devices **12**. When the protocol or client device changes, the layout manager takes care of the necessary adjustments to ensure the GUI components are properly arranged.

The client components **48** may be visual or non-visual components. Visual components include generic GUI components such as windows, buttons and scroll bars. Non-visual components are data managers used to manage the data and values used by the visual components. Non-visual components may also include data tables. It is important to note that the client components are not platform specific and thus a developer does not require knowledge of any of the device characteristics specific to any of the client devices **12**. Due to the generic nature of the client

components **48**, they may be rendered on any of the client devices **12**. The client components can be defined and accessed through the Application Program Interfaces (APIs) **62**, **64**, **66** of the application environment **60**.

The server components **52** are similar in nature to the client components **48**. The server components **52** may be visual or non-visual components. As with the client components **48**, the server components may be visual components including generic GUI components such as windows, buttons and scroll bars and non-visual components such as data managers used to manage the data and values used by the visual components. Non-visual components may also include data tables. Therefore, the server components **52** provide generic, syntax independent interfaces to the server connection devices. Furthermore, the server components **52** may be rendered on any of the servers **36**. Again, the server components 52 can be defined and accessed through the Application Program Interfaces (APIs) **62**, **64**, **66** of the application environment **60**.

The server drivers **54** act as a translator between client devices **12** and the servers **36**. Each type of server 36 in the network requires a different server driver. The driver accepts generic commands from a communicating program and translates them into specialized commands for the particular server. The use of drivers provides the advantage of allowing a high-level programming interface for access to the devices. The developer is not required to delve into the intricacies of communicating with every one of the servers **36**.

The application environment **60** is the API providing an interface to developers to fully utilize the components of

the Maui server **30**. The application environment **60** allows the creation of high-quality, platform-independent graphics (including line art, text, and images) in a single model that uniformly addresses color, spatial transforms, and compositing. Furthermore, the application environment **60** allows developers to access the client drivers **42** and the server drivers **54**, allows developers to access shared resources, and allows developers to access debugging and diagnostics services provided within the application environment **60**.

The application environment **60** further includes three levels of access or usability: basic, advanced and expert. In the basic level, the application environment **60** performs almost all of the underlying details. The layout, platform targeting and almost all of the event handling is performed by the application environment **60**. In the advanced level, the developer must control the layout and arrangement of components and respond various events, resource sharing and server connections. The expert level allows the developer to handle all of the aspects of the device specifics and compositors including event handling, layout and device specific presentation control.

The method of abstracting device independent request-response network interactions involves the translation of client centric request data packages into server centric event object and correspondingly the translation of server centric objects into client centric response data packages.

In general, each client device **12** will have their own native Application Program Interface (API) or Application Program Interfaces (APIs). The API is a set of functions between the application software and the application platform. These functions are a formalized set of software

calls and routines that can be referenced by an application program in order to access supporting system or network services. In this way, the API allows a developer to (a) facilitate communication between a client device and the Maui server **30** and (b) build user interfaces for a client device.

The method of abstracting device independent request-response network interactions is integrated through the application environment **60** of the Maui server **30**. The application environment provides a single Maui API to abstract the client devices' **12** APIs. The Maui API facilitates the creation of the user interface for a client device using a collection of user interface software objects (Maui user interface objects) that represent user interface elements such as text fields, check boxes or buttons. It should be noted that this list is not exhaustive.

The steps in the method of abstracting device independent request-response network interactions will be described with reference to client device **14**. It should be noted that the general steps in the method are essentially the same for all client devices **12** but largely differ in the client device's native protocol.

The client device **14** sends a request to the Maui server **30**. As the client device **14** is a PC running a web browser, the request sent by the client device **14** may be a request for a web page. The Maui server **30** will accept the client device's **14** request in the client device's **14** native protocol. In this particular example, the native protocol will be HTTP where an HTTP command will be sent to the server directing it to fetch and transmit the requested web page. The Maui server **30** will then dynamically translate

the request into Maui Event Objects. The Maui event objects will then be dynamically translated into the server's

The present invention is directed to a method and apparatus for abstracting device-independent request-response network interactions using a high-level application programming interface. native protocol in order to retrieve the requested web page. The requested web page ("the Response") will then be sent from the server **36** to the Maui server **30** in the server's **36** native protocol. The Response will then be dynamically translated into Maui event objects. The Maui event objects manipulate Maui user interface objects, which are then translated into client device **14** specific API calls such as HTML pages. The HTML pages are then communicated to the client device **14** in the client device's **14** native protocol.

Accordingly, while this invention has been described with reference to illustrative embodiments and examples, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments and examples, as well as other embodiments of the invention, will be apparent to persons skilled in the art upon reference to this description. For example, there are numerous possible client devices **12** and specific native protocols for each device.

WHAT IS CLAIMED IS:

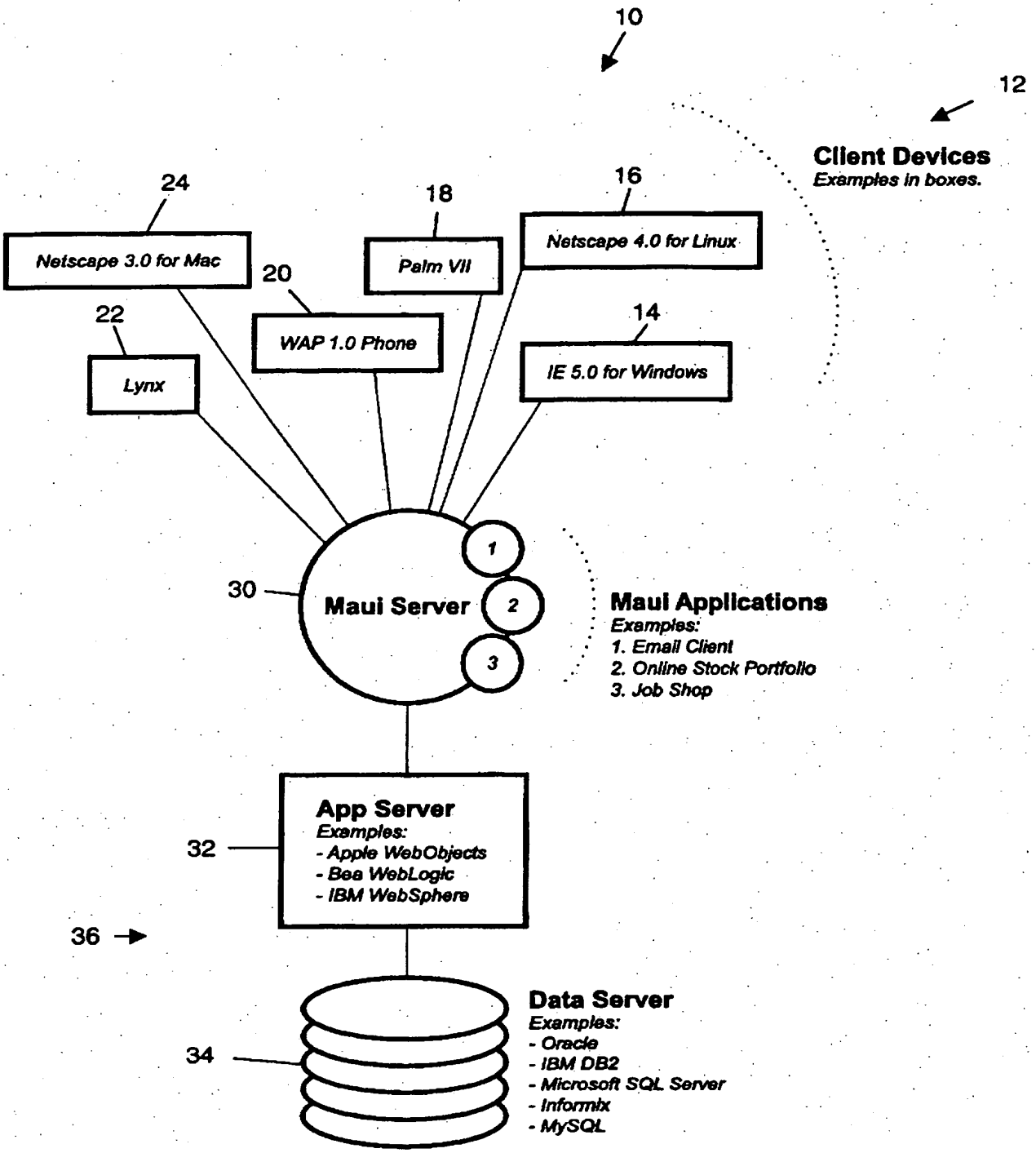

10
12
Client Devices
Examples in boxes.
24
Netscape 3.0 for Mac
18
Palm VII
16
Netscape 4.0 for Linux
20
WAP 1.0 Phone
22
Lynx
14
IE 5.0 for Windows
30
Maui Server
1
2
3
Maui Applications
Examples:
1. Email Client
2. Online Stock Portfolio
3. Job Shop
App Server
Examples:
- Apple WebObjects
- Bea WebLogic
- IBM WebSphere
32
36
Data Server
Examples:
- Oracle
- IBM DB2
- Microsoft SQL Server
- Informix
- MySQL
34

FIG. 1

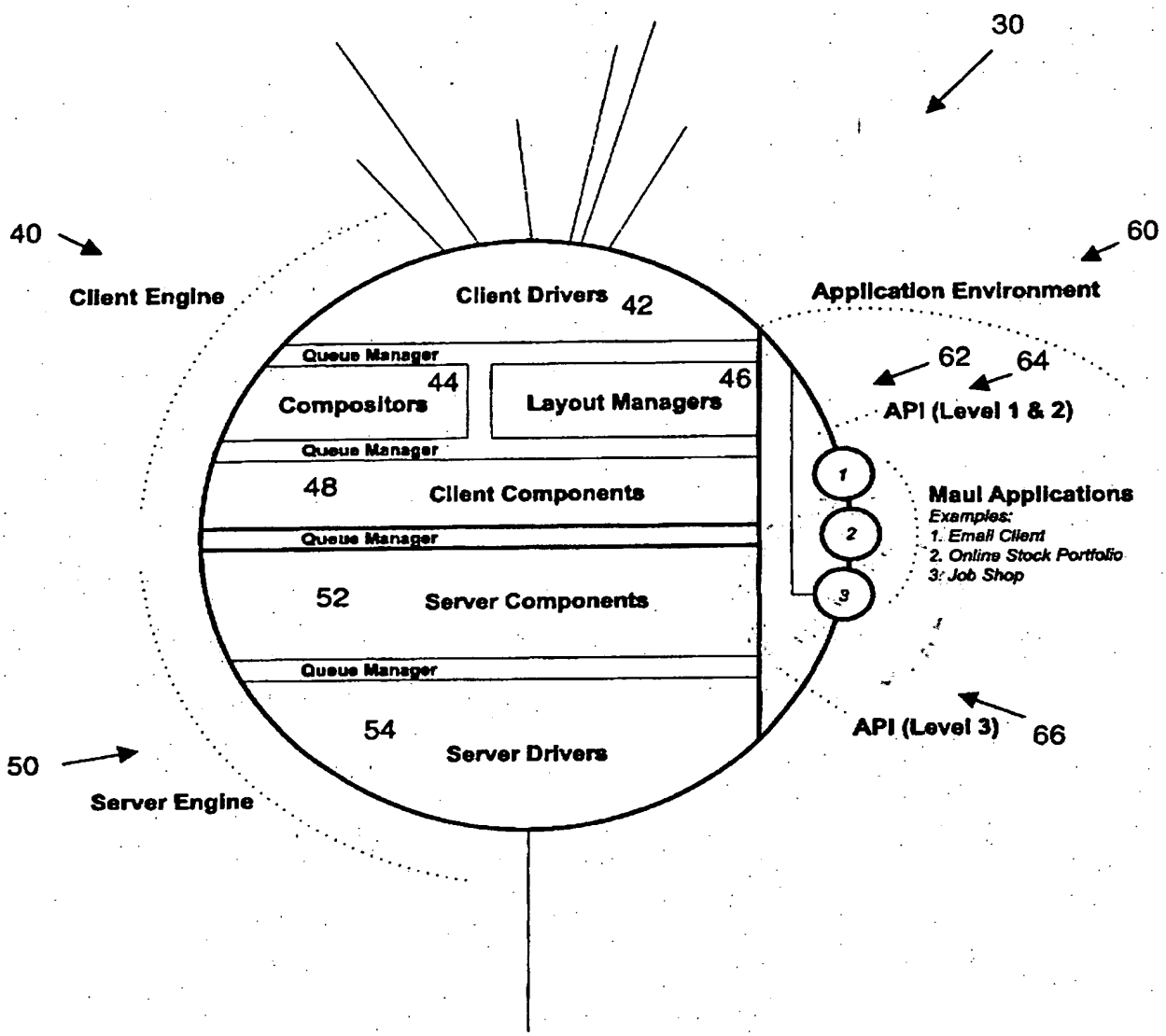

30
40
Client Engine
Client Drivers 42
Queue Manager
Compositors 44
Layout Managers 46
Queue Manager
48 Client Components
Queue Manager
52 Server Components
Queue Manager
54 Server Drivers
50
Server Engine
60
Application Environment
62
64
API (Level 1 & 2)
1
2
3
Maui Applications
Examples:
1. Email Client
2. Online Stock Portfolio
3. Job Shop
API (Level 3)
66

FIG. 2

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [x] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [x] FADED TEXT OR DRAWING
- [ ] BLURED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [x] COLORED OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [ ] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [ ] OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning documents *will not* correct images problems checked, please do not report the problems to the IFW Image Problem Mailbox